Full Stack Web Development Exam

Dear Candidate,

The following exam will provide you with the opportunity to demonstrate your professional skills, ability to work under time constraints, programming experience, and independent problem-solving. The exam is limited to 72 hours from the time of receipt.

Your task is to write a basic functional CRUD application (back-end and front-end) to manage a list of payments and their attributes.

The specifications described below will define the scope of the app.

Your finished app should cover the specifications as best as you can, and you may use whatever tools and references you need, as long as the solutions are viable and explainable by you.

Finally, the finished app code should be pushed to GitHub as a public project. The app should be uploaded/made accessible remotely, and a working link along with the GitHub link should be sent to vadim@adcore.com.

Good luck!

Specification

I. Download CSV file representing payment data:

Use the provided sample file (attached as payment_information.csv). It contains initial data to understand the schema of the Payment object:
payee_first_name - text
payee_last_name - text
payee_payment_status – text (completed, due_now, overdue, pending)
payee_added_date_utc - UTC timestamp
payee_due_date – date - YYYY-MM-DD
payee_address_line_1 – text (mandatory)
payee_address_line_2 - text
payee_city – text (mandatory)
payee_country – text (mandatory, ISO 3166-1 alpha-2)
payee_province_or_state – text - (optional)
payee_postal_code – text - (mandatory)
payee_phone_number – text – (mandatory, E.164)
payee_email – text - (mandatory)
currency – text – (mandatory, ISO 4217)
discount_percent - number (optional, percentage, 2 decimal points)
tax_percent - number (optional, percentage, 2 decimal points)
due_amount – number (mandatory, 2 decimal points)
total_due – calculated number (2 decimal points)

II. Build the back-end using Python 3+ and FastAPI

- Normalize the CSV data using Pandas, ensuring all fields conform to the schema.
- Save the normalized data into MongoDB.
- Implement file upload functionality:
  - Allow users to upload evidence files (PDF, PNG, JPG) when the payment status is updated to “completed”.
  - Payments cannot be marked as completed without uploading an evidence file.
  - Store the evidence file in MongoDB.
  - Generate a download link for the uploaded evidence file.
- Create the following web services:
  - get_payments(...): Fetches payments with the following calculations performed server-side:
    Change the payee_payment_status to “due_now” if payee_due_date is today.
    Change the payee_payment_status to “overdue” if payee_due_date smaller than today.
    Calculate total_due based on discount, tax and due_amount
    Method should support filter, search and paging
  - update_payment(...): Updates one payment.
  - delete_payment(...): Deletes one payment by ID. Returns success or error.
  - create_payment(...): Creates a new payment. Returns the ID of the new record or error.
  - upload_evidence(...): Allows uploading evidence files (PDF, PNG, JPG) when updating status to completed.
  - download_evidence(...): Returns uploaded evidence file which should be saved from UI.

III. Build the front-end using Angular 15+

Create a Payment Management UI where users can:

- Search for payments with filters (on text fields).
- Display total_due and valid payment status per row calculated dynamically on the server.
- Ensure seamless server-side pagination.
- View payment details.
- Update payments (only due_date, due_amount, and status are editable).
- Upload evidence when changing status to completed. Show validation error if no evidence is uploaded.
- Provide a download link for evidence if available.
- Delete payments.
- Add new payments using a form.
- List of mandatory screens/dialogs
  Main Screen:
  - Display total_due calculated dynamically on the server.
  - Filter and display payment details, including due_now status.
  - Include an option to upload evidence for completed payments.
  - Provide a download link for the evidence file if it exists.

  Add Payment Screen:
  - Include address and currency auto-complete.
  - Fields should have appropriate validations.
  - Status can be pending only

  Edit Payment Screen:
  - Include address and currency auto-complete.
  - Fields should have appropriate validations.
  - Status can be pending, due_now, completed
  - Prevent changing to completed without uploading evidence.

Use external APIs to enhance the user experience:

- Auto-complete fields for country, city, state and currency using any free API like https://countriesnow.space/.
- Use the same API to validate and extend city and country with connected selectors .
- Implement pagination with server-side support.
- Validate user input and show errors when necessary.
- Format payee_added_date_utc in the UI to show local date and time in the format: Jan 10, 2024, 3:00 PM.
- Use third-party libraries for UI elements like Autocomplete, Calendar, Tooltips, Loading Bars, etc. (Choose from Material, PrimeNG, or ng-bootstrap).

IV. Deployment

- Deploy your finished application to a cloud/hosting solution (e.g., Heroku, Azure, AWS) or use ngrok to share your local environment.
- Push your code to a public GitHub repository.
- Submit the working application link and GitHub repository to vadim@adcore.com.

Good luck!